ELSEVIER

Discrete Applied Mathematics 72 (1997) 155–166

DISCRETE APPLIED MATHEMATICS

Scheduling with forbidden sets

Markus W. Schäffter *

Fachbereich Mathematik, MA 6-1, Technische Universität Berlin, Straße des 17, Juni 135, 10623 Berlin, Germany

Received 17 December 1993; revised 26 March 1995

Abstract

We consider the case of minimizing the makespan when scheduling tasks with respect to a class of so-called forbidden sets, i.e. sets of tasks that are not allowed to be scheduled in parallel. Following the notion of Graham et al. [7], the treated problem will be abbreviated by $P\infty|FS|C_{\max}$. This problem is $\mathcal{NP}$ hard even for unit task length and forbidden sets that contains exactly two elements. Moreover, we show that the existence of a polynomial-time approximation algorithm with a worst-case ratio strictly less than 2 implies $\mathcal{P} = \mathcal{NP}$. We point out that the corresponding re-optimization problem (after changing the instance slightly) is $\mathcal{NP}$ hard as well, even if only one forbidden set is added or removed. Furthermore, the latter re-optimization problems have approximation thresholds of 3/2 and 4/3, respectively. To conclude our results, we show that the bound of 3/2 is tight for the case of re-optimizing an *optimal schedule* after adding a new forbidden set.

Keywords: Scheduling; Forbidden sets; Approximation; Approximation threshold; m-Machine problem; Sensitivity

1. Introduction

Forbidden Sets are a powerful tool for describing "compatibility constraints" within complex problems. A common example for this is scheduling tasks on different types of machines. Consider the following situation. There are n tasks requiring the same type of machine, and there exist exactly k machines of this type. Then, at most k of these n tasks could be scheduled in parallel at any time. The common way modelling this situation is by so-called *forbidden sets*, i.e. sets of tasks that are not allowed to be scheduled in parallel. A detailed survey on forbidden sets can be found in [1, Section 2]. For the given example, take as forbidden sets all sets containing exactly $k + 1$ of the n given tasks to ensure that at most k tasks can be scheduled in parallel at any time.

The problem of scheduling tasks with respect to forbidden sets is a generalization of the *m-machine problem* (*without precedence constraints*). In Section 2 we show

* E-mail: shefta@math.tu-berlin.de.

PII S0166-218X(96)00042-X

that the discussed problem of scheduling tasks with respect to forbidden sets remains $\mathcal{NP}$ complete even for unit task length, forbidden sets of size two, and no precedence constraints involved.

An interesting question is whether a "good" solution can be maintained under small changes. Tovey [10] gave three example instances for the *m-machine problem* (*without precedence constraints*) which show that a list of tasks determining an optimal list schedule on m identical machines can produce a list schedule on m' (with $|m - m'| = 1$) identical machines that is 50% away from optimality. Thus, list schedules are very sensitive with respect to changes of the instance. On the other hand, the *largest processing time first* (LPT) heuristic computes schedules with a relative error bounded by $1/3 - 1/(3m)$ for any number m of machines [6]. So in this case better newly construct a schedule after a change of the instance.

For the more general case of *scheduling with respect to forbidden sets*, we show that the existence of any polynomial-time algorithm that determines schedules of length strictly less than 2 times the makespan implies $\mathcal{P} = \mathcal{NP}$.

In Sections 3 and 4, we consider the problem of re-optimization after adding or removing one forbidden set. These problems are relevant for interactive optimization, i.e. optimizing step-by-step, adding in each step additional constraints to find a schedule that fits the real-life demands as much as possible (cf. [1,2] for more details). We point out that these re-optimization problems are also $\mathcal{NP}$ hard and that they have approximation thresholds of 3/2 and 4/3, respectively. To conclude this paper, we present a polynomial-time approximation algorithm for the case of re-optimizing an *optimal schedule* after adding a forbidden set. This algorithm yields a bound of exactly 3/2 times the makespan and hence is optimal under all polynomial-time approximation algorithms for the problem of re-optimizing after adding a new forbidden set.

2. Preliminaries

Given a set $V = \{v_1, v_2, \ldots, v_n\}$ of tasks and a positive integer length $l(v)$ for each task $v \in V$. A *schedule of the tasks in* V is a vector $(t_1, \ldots, t_n)$ of positive integers. We call t_i the *starting time of task* v_i. For shortness, we denote the starting time of a task $v \in V$ by $t(v)$ if necessary. Similarly, we call $t(v) + l(v)$ the *finishing time of the task* v. The *length of a schedule* is the latest finishing time among all tasks, i.e. $\max\{t(v) + l(v) | v \in V\}$. A schedule is called *feasible with respect to the precedence order* $\Theta = (V, \prec)$ if for all tasks u and v with $u \prec v$, the task v is started after finishing task u, i.e. $v \succ u$ implies $t(v) \geqslant t(u) + l(u)$.

For a scheduling problem, we denote the minimal length of a feasible schedule for a given instance I by *makespan*(I) or *makespan* for short. We say an approximation algorithm has a *worst case ratio* of $c \geqslant 1$ for all given instances I, it produces a schedule of length *length*(I) fulfilling *length*(I)/*makespan*(I) $\leqslant c$. We call c an *approximation threshold* if the existence of a polynomial-time approximative algorithm with a worst

case ratio less or equal to c would imply $\mathscr{P} = \mathscr{NP}$ (see Section 4 and [4] for more details).

In the case of scheduling with respect to forbidden sets, we introduce a class $\mathscr{F}$ of subsets of V, so-called *forbidden sets*. A schedule is called *feasible with respect to* $\mathscr{F}$ if for all forbidden sets $F \in \mathscr{F}$ not all tasks of F are scheduled in parallel at any time unit. In the following, we call a schedule *feasible* for short if it is feasible with respect to the considered class $\mathscr{F}$ of forbidden sets.

Scheduling with respect to Forbidden Sets

Instance: Given a finite set $V = \{v_1, v_2, \ldots, v_n\}$ of tasks, a positive integer length $l(v)$ for all tasks $v \in V$, a family $\mathscr{F} = \{F_1, F_2, \ldots, F_s\}$ of subsets F_i of V, and a deadline D.

Question: Does there exist a schedule of length at most D such that for all forbidden sets $F \in \mathscr{F}$, not all tasks of F are scheduled in parallel at any time?

This problem is $\mathscr{NP}$ complete in the strong sense since it includes the *m-machine problem* (*without precedence constraints*) with $m \geqslant 2$ fixed. (For the transformation, set $\mathscr{F} = \{F \subseteq V \mid |F| = m + 1\}$ to ensure that at most m tasks can be scheduled in parallel.) In the following we show that the problem remains $\mathscr{NP}$ complete even for unit task length and forbidden sets of size two. Note that we have defined the above problem such that no precedence constraints are involved.

The Partition Problem

Instance: A finite set A and a strictly positive integer size $l(a)$ for each $a \in A$.

Question: Is there a subset $A' \subseteq A$ such that $\sum_{a' \in A'} l(a') = \sum_{a \in A - A'} l(a)$?

The Set Splitting Problem

Instance: Given a collection $\mathscr{C}$ of subsets of a finite set S.

Question: Does there exist a partition of S into two subsets S_1 and S_2 such that no subset in $\mathscr{C}$ is entirely contained in either S_1 or S_2?

Both problems, the *Partition Problem* as well as the *Set Splitting Problem* are $\mathscr{NP}$ complete (in the weak and in the strong sense, respectively). The latter problem remains $\mathscr{NP}$ complete in the strong sense even if all $C \in \mathscr{C}$ have at most 3 elements (see [4,9]).

The Graph K-Colourability Problem

Instance: Given an undirected graph $G = (V, E)$ and a positive integer $K \leqslant |V|$.

Question: Is G K-colourable, i.e. does there exist a function f assigning each vertex $v \in V$ a colour of $\{1, 2, \ldots, K\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

The Graph K-Colourability Problem is solvable in polynomial time for $K = 2$, but remains $\mathscr{NP}$ complete for all fixed $K \geqslant 3$ (see [4]). It is also $\mathscr{NP}$ complete for

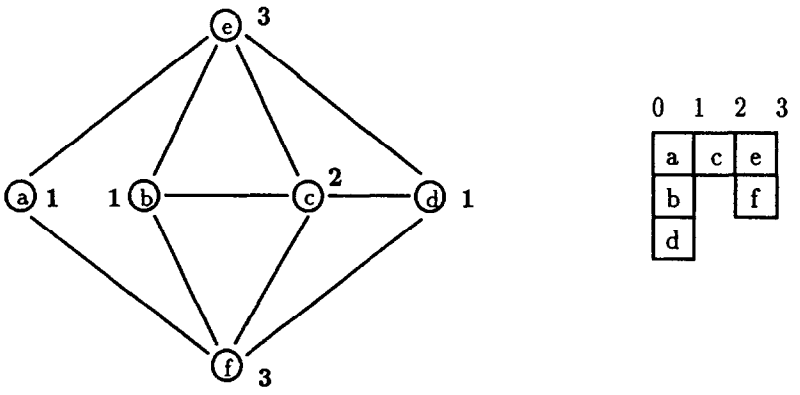

Fig. 1. A graph colouring with 3 colors and the corresponding schedule.

$K = 3$ and planar graphs having no vertex degree exceeding 4 [5]. The corresponding optimization problem is to find a legal colouring with a minimal number of different colours for a given undirected graph $G = (V,E)$. We call this optimization problem the *Optimal Graph Colouring Problem.*

As we will see, the Graph K-Colourability Problem can be formulated as a special case of the problem Scheduling with respect to Forbidden Sets. Hence, the problem Scheduling with respect to Forbidden Sets is $\mathcal{NP}$ complete for any deadline $D \geqslant 3$ and unit task length.

Lemma 1. *The Graph K-Colourability Problem is equivalent to the problem Scheduling with respect to Forbidden Sets in which all tasks have unit length and each forbidden set $F \in \mathcal{F}$ contains exactly two elements, i.e. both problems can be polynomially transformed into each other.*

Proof. Consider an instance $(G = (V,E),K)$ of the Graph K-Colourability Problem. We will now transform this instance to an instance of the problem Scheduling with respect to Forbidden Sets as follows.

- Introduce for each vertex v in the graph G a task v of length 1.
- Set the class $\mathcal{F}$ of forbidden sets to the set of edges in the graph G, i.e. $\mathcal{F} = E$.
- Set the deadline $D = K$.

Using this transformation, each schedule of length $k \leqslant K$ that is feasible with respect to $\mathcal{F}$ induces a colouring of the vertices of G with exactly k different colours and vice versa. The colour of a vertex corresponds to the finishing time of the corresponding task. There is a one-to-one correspondence between feasible colourings and feasible schedules for instances where all tasks have unit length and every forbidden set $F \in \mathcal{F}$ contains exactly two elements. □

In the next section, we use the following fact. There is a linear-time algorithm that produces a colouring with at most $\Delta(G)$ (the maximum vertex degree in G) different

colours for all connected graphs G that are neither complete graphs nor Odd Cycles in the following way: Given a "proper" order $v_1,\ldots,v_n$ on the vertices, proceed step-by-step, colouring in the ith step vertex v_i with the lowest colour $1,\ldots,k$ that is not possessed by its already coloured neighbours. Such a "proper" order can be obtained by recursively removing a vertex of minimum degree. (See [8, Appendix B, p. 380f] or [3, Chapter V, p. 99, Exercise 3]).

Since Odd Cycles and complete graphs (with $\Delta(G) \leqslant 4$) can easily be detected and coloured with at most 4 different colours, this algorithm can be extended to arbitrary graphs G with $\Delta(G) \leqslant 4$.

3. Sensitivity analysis

Assume that a feasible schedule is known for a fixed instance I of the problem Scheduling with respect to Forbidden Sets. It is advantageous if this schedule is optimal or nearby the optimum, but for the following results it is sufficient to have an arbitrary feasible schedule. Does this knowledge help to determine a feasible schedule for an instance I' of the same problem that differs only slightly from the fixed one, i.e. has one forbidden set more or less? The answer to this question is relevant for online-optimization, i.e. optimizing step-by-step, extending in each step the knowledge about the problem instance.

3.1. Adding a forbidden set

Adding a new forbidden set can affect the feasibility and/or the optimality of a given schedule. But how difficult is it to reestablish feasibility if the added forbidden set is violated by the given schedule?

Update After Adding a Forbidden Set

Instance: Given a feasible schedule of length L for a fixed instance I of the problem Scheduling with respect to Forbidden Sets and a new forbidden set $F' \notin \mathcal{F}$.

Question: Does there exist a schedule of length at most L such that for all forbidden sets $F \in \mathcal{F}' = \mathcal{F} \cup \{F'\}$, not all tasks of F are scheduled in parallel at anytime?

Note that a feasible schedule will be even optimal for the new instance I' if the given schedule is optimal for I.

Lemma 2. *The decision problem Update After Adding a Forbidden Set is $\mathcal{NP}$ complete in the ordinary sense even if the added forbidden set F contains only two elements.*

Proof. The proof is done by a transformation from the *Partition Problem* to the problem *Update After Adding a Forbidden Set.*

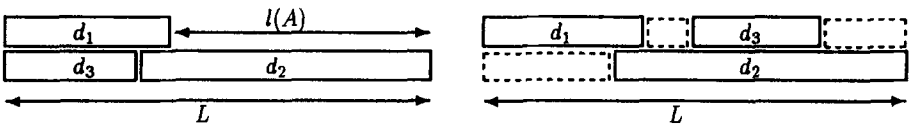

Fig. 2. An optimal schedule and one holding the forbidden set $\{d_1, d_3\}$.

- Let $(A, l(.))$ be an instance of the *Partition Problem.*
- Set $l(A) = \sum_{a \in A} l(a)$.
- Introduce a task a of length $l(a)$ for each $a \in A$.
- Define three dummy tasks d_1, d_2 and d_3 of length $l(A)/2 + 1$, $l(A) + 1$ and $l(A)/2$, respectively.
- The family $\mathcal{F}$ is set to all subsets of $\{d_1, d_2, d_3\} \cup A$ with 3 elements plus all sets $\{d_3, a\}$ with $a \in A$.
- Set the length to $L = 3/2\ \ l(A) + 1$.

The resulting instance I of the problem Scheduling with respect to Forbidden Sets can easily be solved within the given schedule length L. Since the class $\mathcal{F}$ includes all sets containing 3 elements, at most 2 tasks can be scheduled in parallel at any time unit. The left hand side of Fig. 2 shows the shape of a possible feasible schedule. The tasks d_3 and d_2 are scheduled one after the other on the second machine, while all tasks $a \in A$ can be scheduled in sequence behind the task d_1 on the first machine. Since at any time unit, exactly two tasks are scheduled in parallel, the schedule cannot be shortened, i.e. it is of optimal length $L = makespan(I)$.

Adding the set $F' = \{d_1, d_3\}$ to the class $\mathcal{F}$ of forbidden sets determines another instance I' to the problem Scheduling with respect to Forbidden Sets. The new forbidden set F does not allow the tasks d_1 and d_3 to be scheduled in parallel. There is no way to schedule the tasks d_1 and d_3 on different machines without exceeding L. (Scheduling d_2 before d_3 on the second machine will lengthen the schedule since the task d_3 cannot be scheduled in parallel with any task a in A.)

Thus, the tasks d_1 and d_3 have to be scheduled on the same machine. Fig. 2 shows on the right hand side the shape of the resulting schedule. The tasks d_1, d_2, d_3 cause two slots, one on every machine and each of length $l(A)/2$. Because of this, each feasible schedule divides the set of tasks into two subsets A' and $A - A'$ of equal length. Hence, the question whether there exists a feasible schedule solves the given instance $(A, l(.))$ of the Partition Problem. Note that each slot may be divided into multiple parts by d_1, d_3 and d_2, respectively. □

3.2. *Removing a forbidden set*

Removing a forbidden set keeps the schedule feasible. To determine an optimal or even a "good" schedule for the resulting instance, does it help to have an optimal schedule for the original instance?

Update After Removing a Forbidden Set

Instance: Given a feasible schedule of length L for an instance I of the problem Scheduling with respect to Forbidden Sets, a forbidden set $F' \in \mathcal{F}$ and a positive integer $L' < L$.

Question: Does there exist a schedule of length at most L' such that for all forbidden sets $F \in \mathcal{F}' = \mathcal{F} \setminus F'$, not all tasks of F are scheduled in parallel at any time?

Lemma 3. *The decision problem Update After Removing a Forbidden Set is $\mathcal{NP}$ complete in the strong sense even if all tasks have unit length, all forbidden sets $F \in \mathcal{F}$ contain exactly two elements, and $L' = L - 1$.*

Proof. A smaller result can be proved using the same transformation technique as in Section 3.1. To obtain the above result, we need a transformation from a special case of the Graph K-Colourability Problem to the problem Scheduling with respect to Forbidden Sets. It is known that the Graph K-Colourability Problem remains $\mathcal{NP}$ complete for fixed $K = 3$ on planar graphs with a maximal vertex degree of 4 (see [4]). Let us consider without loss of generality only instances $(G = (V,E))$ of this special problem. We now transform those instances to instances of the problem Update After Removing a Forbidden Set.

- Let $G = (V,E)$ be the given instance of the Planar Graph 3-Colourability Problem, i.e. a planar undirected graph of maximal vertex degree $\Delta(G) \leqslant 4$.
- Extend G to a new planar graph $G' = (V',E')$ as follows. Take a complete graph K_4 with $V = \{v_1,\ldots,v_4\}$ and append it to any vertex $v \in V$ by including a new edge: $E' = E \cup \{\{v,v_1\}\}$.
- Introduce a task v of unit length for each $v \in V'$.
- Set $\mathcal{F} = E'$ and $L = 4$.

To obtain an instance of the problem Update After Removing a Forbidden Set, we must find a feasible schedule for the instance resulting from the transformation. This can be done by constructing a feasible 4-colouring for the extended graph G' in the following way. First apply the colouring algorithm presented in Section 2 to the graph G. This determines a feasible 4-colouring for the subgraph G of G' within polynomial time since $\Delta(G) \leqslant 4$. Then the vertices $v_1,\ldots,v_4$ can be coloured with the same four colours such that v and v_3 have different colours.

Since G' contains at least one subgraph isomorphic to the graph K_4 (the subgraph induced by $v_1,\ldots,v_4$), it cannot be coloured with less than 4 different colours. Thus, the determined 4-coloring is optimal for the graph G' and hence is the schedule of length 4 induced by this colouring too. Setting $F' = \{v_3,v_4\}$ will cause an instance of the problem Update After Removing a Forbidden Set.

Note that the graph $G'' = (V,E' \setminus \{v_3,v_4\})$ is 3-colourable if and only if G is 3-colourable since in G'' the graph induced by $v_1,\ldots,v_4$ can be coloured with three colors. Thus solving the above instance of the update problem solves the $\mathcal{NP}$ complete Planar Graph 3-Colouring Problem for the given instance $G = (V,E)$. □

4. On the nonexistence of good solutions

As we have seen in Section 3, the problem Scheduling with respect to Forbidden Sets and even its update versions are $\mathcal{NP}$ complete. The following lemmas tell us that we cannot expect the existence of good polynomial-time approximation algorithms for the corresponding optimization problems, unless $\mathcal{P} = \mathcal{NP}$.

Lemma 4. *If $\mathcal{P} \neq \mathcal{NP}$, then no polynomial-time approximation algorithm for the optimization problem of Scheduling with respect to Forbidden Sets can have a worst case ratio strictly less than* 2. *This remains true, even if all tasks have unit length and every forbidden set contains exactly two elements.*

Proof. Garey and Johnson [4, Theorem 6.11, p. 144] have shown that 2 is an approximation threshold for the Optimal Graph Colouring Problem. As we have seen in Section 2, each polynomial-time approximation algorithm for the optimization problem of Scheduling with respect to Forbidden Sets implies a polynomial-time approximation algorithm for the Optimal Graph Colourability Problem of the same worst case ratio. Thus, 2 also is an approximation threshold for the optimization problem Scheduling with respect to Forbidden Sets as well. □

As we will see in the following, similar statements can be given for the update versions of the problem Scheduling with respect to Forbidden Sets.

Lemma 5. *If $\mathcal{P} \neq \mathcal{NP}$, then no polynomial-time approximation algorithm for the optimization version of the problem Update After Adding a Forbidden Set can have a worst case ratio strictly less than* 3/2. *This remains true, even if all tasks have unit length and the given schedule is required to be optimal and not only feasible.*

Proof. The proof is done by a transformation from the Set Splitting Problem. Assume that there exists a polynomial-time approximation algorithm A determining a schedule of length strictly less than 3/2 times the makespan for any instance I' resulting from an instance I by adding a new forbidden set, if a feasible schedule for the primary instance I is given.

Let $(S, \mathcal{C} = \{C_1, \ldots, C_k\})$ be an instance of the Set Splitting Problem. Consider that $|C_i| \leqslant 3$ for all $C_i \in \mathcal{C}$. This can be done without loss of generality since the Set Splitting Problem remains $\mathcal{NP}$ complete in the strong sense even if all $C \in \mathcal{C}$ have at most 3 elements (see [4,9]).

We will find the solution for this instance running the approximation algorithm A at most $k = |\mathcal{C}|$ times, beginning with $I_1 = (S, \mathcal{F} = \{C_1\})$ as an initial instance. For I_1, a feasible schedule of length 2 can be constructed in the following way. Schedule all tasks contained in C_1 at time unit 1 with the exception of an arbitrary task $x_1 \in C_1$ which is scheduled at time unit 2. Moreover, this schedule is optimal as well.

The given instance of the Set Splitting Problem is a "YES" instance if and only if the corresponding instance of the scheduling problem allows a feasible schedule of length 2. Adding the set C_2 to the class $\mathcal{F}$ of forbidden sets gives a new instance $I_2 = (S, \mathcal{F} = \{C_1, C_2\})$ for the scheduling problem. Since we already know a feasible schedule for the instance I_1, algorithm A computes a feasible schedule of length $L(I_2)$ strictly smaller than 3/2 times *makespan* (I_2).

If $L(I_2) \geqslant 3$, the makespan of I_2 must be strictly greater than $3/(3/2) = 2$, which means that the tasks in S cannot be scheduled within two time units with respect to $\mathcal{F} = \{C_1, C_2\}$. This implies that the underlying instance of the Set Splitting Problem does not allow a splitting into two subsets. On the other hand, if the resulting schedule has length 2, the set S can be split into two subsets $S_1 = \{$all tasks scheduled on the first time unit$\}$ and $S_2 = S \setminus S_1$ with respect to the sets C_1 and C_2. Note that all schedules of length 2 have to be optimal since already C_1 causes two time intervals in any feasible schedule.

Thus, as long as the resulting schedule is of length 2, we add another set C_i to $\mathcal{F}$, applying algorithm A to the resulting instance. If this can be continued till $\mathcal{F} = \mathcal{C}$ and the resulting schedule is of length 2, a splitting (S_1, S_2) is found for the given instance $(S, \mathcal{C})$ of the Set Splitting Problem. Let $S_1 = \{$all tasks scheduled on the first time unit$\}$ and $S_2 = S \setminus S_1$. On the other hand, as soon as the algorithm A determines a schedule of length greater or equal to 3, the set S cannot be split into two subsets with respect to the current class $\mathcal{F} = \{C_1, \ldots, C_i\}$, implying that S cannot be split with respect to $\mathcal{C}$ either.

So after at most k calls of algorithm A, we know whether S can be split into two subsets with respect to $\mathcal{C}$ or not. Note that k is part of the input $(S, \mathcal{C})$ and $k \leqslant \binom{n}{3} \in \mathcal{O}(n^3)$. Assuming that A runs in polynomial time, we have found a way solving the Set Splitting Problem in polynomial time. Thus, the existence of such a polynomial-time algorithm A would imply $\mathcal{P} = \mathcal{NP}$. □

Lemma 6. *If $\mathcal{P} \neq \mathcal{NP}$, then no polynomial-time approximation algorithm for the optimization version of the problem Update After Removing a Forbidden Set can have a worst case ratio strictly less than* 4/3. *This remains true, even if all tasks have unit length and the given schedule has to be optimal and not only feasible.*

Proof. Assume that there exists a polynomial-time approximation algorithm for the problem Update After Removing a Forbidden Set with a worst case ratio strictly less than 4/3. Apply this algorithm to the instance constructed in Section 3.2. The resulting schedule has length 3 if and only if the graph G of the underlying instance of the Planar Graph 3-Colouring Problem is 3-colourable. Thus applying the algorithm to the above instance would solve the $\mathcal{NP}$ complete Planar Graph 3-Colouring Problem for the given graph $G = (V, E)$. □

As we have seen, there cannot exist a polynomial-time approximation algorithm for the problem Update After Adding a Forbidden Set with a relative error strictly less than

50%, unless $\mathscr{P} = \mathscr{NP}$. In particular, this is true for arbitrary task lengths, involved precedence constraints and if the given schedule is optimal.

To conclude, we give a polynomial-time approximation algorithm for the problem of extending the system of forbidden sets if an *optimal schedule* is given. This algorithm works for arbitrary task lengths and so in particular for unit task lengths but not for precedence constraints. Thus, the approximation threshold of 3/2 is tight for this case.

Algorithm

Given an *optimal schedule* (of length $L = makespan(I)$) for a fixed instance $I = (V, \mathscr{F})$ of the problem Scheduling with respect to Forbidden Sets. Let $I' = (V, \mathscr{F}' = \mathscr{F} \cup \{C'\})$ be a further instance caused by adding a new forbidden set C' to the class $\mathscr{F}$ of forbidden sets. The following instructions compute a feasible schedule for the instance I' of length $L' \leqslant 3/2\ makespan(I')$.

(1) If C' is not violated by the given schedule, STOP. In this case, the given schedule is also optimal for the new instance I'.

(2) Otherwise, there is a time unit when all tasks contained in C' are scheduled in parallel in the given schedule. Shift the shortest task v' of C' behind the end of the given schedule such that v' is scheduled after all the other tasks. The resulting schedule obviously respects the forbidden set C' as well as all the other forbidden sets in $\mathscr{F}$.

Lemma 7. *Given an optimal schedule for the instance I, the above algorithm produces in polynomial time a feasible schedule for the instance I' whose length L' is at most* 3/2 *times the makespan of I'.*

Proof. First of all, the algorithm checks, whether the added forbidden set C' is violated by the given schedule or not. If it is not violated, the given schedule remains optimal. Otherwise, one task v' will be shifted to the end of the schedule, lengthening the schedule by the amount of $l(v')$ (Fig. 4). It is obvious that the algorithm runs in polynomial time and that the computing time is bounded by $\mathcal{O}(|V|)$ if the schedule is represented by the starting times and the task lengths.

Then, the length of the resulting schedule is $L' = L + l(v')$. Note that in the resulting schedule, at least two jobs of the added forbidden set F have to be scheduled one after the other. This means $makespan(I') \geqslant l(v') + l(v'')$. Since v' is the shortest task of C', $l(v') \leqslant 1/2\ makespan(I')$. Now, $L \leqslant 3/2\ makespan(I')$ follows from $L' = L + l(v')$ with $L = makespan(I) \leqslant makespan(I')$. □

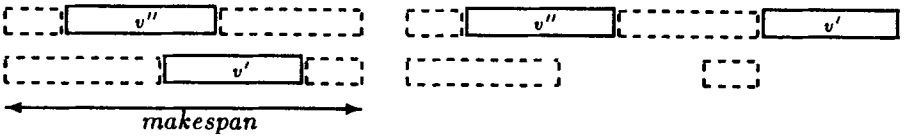

Fig. 3. A shift step reestablishes the feasibility.

5. Concluding remarks

The result of Lemma 7 can easily be extended.

- If the given schedule guarantees $L \leqslant c \cdot makespan(I)$ for some constant $c \geqslant 1$, we obtain $L' \leqslant (c + 1/2)\, makespan(I')$ for the length L' of the resulting schedule.
- Moving one of the tasks of the added forbidden set F to the end of the schedule is not the only way to hold F. Another way is the following. Let v' and v'' be the two shortest tasks in F and assume without loss of generality that $t(v'') \leqslant t(v')$. Then, schedule all tasks starting after v'' has finished (tasks v with $t(v) \geqslant t(v'') + l(v'')$) exactly $t(v') - t(v'')$ time units later. This approach has the advantage that it keeps existing precedence constraints feasible. Thus, this algorithm also works for scheduling problems with forbidden sets and precedence constraints.
- Moreover, the algorithm maintains assignments of jobs to a special (class of) machine(s).
- The algorithm gives a guaranty for the quality of the solution in terms of *schedule-length* $\leqslant c \cdot makespan(I)$ for the given instance I. Thus this approach can be used for interactive optimization. Starting with an initial (possibly void) instance I_0 and a feasible schedule for this instance with a guaranty $length \leqslant c_0\, makespan(I_0)$, change in each step the instance I_k by adding/removing or changing one parameter such as task-length, precedence constraints, forbidden sets, or a job-to-machine assignment. First, start the presented algorithm to reestablish feasibility. This also gives a new guaranty $length \leqslant c_{k+1}\, makespan(I_{k+1})$. To improve this bound, apply some heuristics like local search, genetic algorithms, etc. Even if these heuristics does not give general guarantees for the quality of the resulting schedule, we can now give a guaranty for the very special instance I_{k+1} as follows. Suppose that applying the mentioned heuristics will shorten the given schedule by a factor of $\alpha < 1$. Then, the resulting schedule holds $length \leqslant (\alpha \cdot c_{k+1})\, makespan(I_{k+1})$. This procedure can be iterated as long as the produced schedule does not fit the demands.

While for the case of a restricted number of machines, the LPT-strategy determines very good suboptimal schedules, a similar approximation algorithm does not exist for the more general case of forbidden sets. Two is an *approximation threshold* for the problem Scheduling with respect to Forbidden Sets. It is still an open question whether this bound is tight or if there exist even larger approximation thresholds. An interesting heuristic determining a feasible schedule is the following. It computes a feasible schedule time-unit by time-unit, using in each step the following priority rule.

Schedule all tasks time-unit by time-unit, beginning each column $[t, t+1]$ with an arbitrary task, adding step by step further tasks until there is no other task which can be added without violating at least one forbidden set. Thereby, choose under all feasible tasks one task v which minimizes the cardinality of the set $C_t \cap C_v$ where C_t denotes the forbidden sets that contain a tasks scheduled during $[t, t+1]$ and C_v denotes the forbidden sets that contain task v.

Considering the corresponding re-optimization problems shows the same difficulties. Both re-optimization problems are $\mathcal{NP}$ hard even if only one forbidden set is added or removed. Both problems hold similar approximation thresholds but we know that 3/2 is a tight bound for the problem Update After Adding a Forbidden Set.

Acknowledgements

The author thanks his supervisor, Rolf Möhring, and his colleagues and the referees for the detailed hints which improved the quality of this paper.

References

[1] M. Bartusch, R.H. Möhring and F.J. Radermacher, Scheduling project networks with resource constraints and time windows, Ann. Oper. Res. 16 (1988) 201–240.
[2] M. Bartusch, R.H. Möhring and F.J. Radermacher, A conceptional outline of a DSS for scheduling problems in the building industry, Decision Support Systems 5 (1989) 321–344.
[3] B. Bollobás, Graph Theory, An Introductory Course (Springer, Berlin, 1979).
[4] M.R. Garey and D.S. Johnson, Computers and Intractability, A Guide to the Theory of $\mathcal{NP}$-Completeness (Freeman, San Francisco, 1979).
[5] M.R. Garey and D.S. Johnson and L. Stockmeyer, Some Simplified $\mathcal{NP}$-Complete Graph Problems, Theor. Comput. Sci. 1 (1976) 237–267.
[6] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell System Technical J. 45 (1986) 1563–1581.
[7] R.L. Graham, E.L. Lawler and J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[8] D. Jungnickel, Graphen, Netzwerke und Algorithmen [German], (BI Wissenschaftsverlag, 2. revised edition, 1990).
[9] L. Lovász, Covering and coloring of hypergraphs, in: Proc. 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing (Utilitas Mathematica Publishing, Winnipeg, 1973) 3–12.
[10] C.A. Tovey, Rescheduling to minimize makespan on a changing number of identical processors, Naval Res. Logist. Quat. 33 (1986) 717–724.